AD-757 983

NETGEN: A PROGRAM FOR GENERATING LARGE SCALE (UN) CAPACITATED ASSIGNMENT, TRANSPORTATION, AND MINIMUM COST FLOW NETWORK PROBLEMS

Darwin Klingman, et al

Texas University

Prepared for:

Office of Naval Research

February 1973

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

DDC
RECEIVED
APR 6 1973
B

Approved for public release;
Distribution Unlimited

Research Report
CS 109

NETGEN

A PROGRAM FOR GENERATING LARGE SCALE (UN)CAPACITATED ASSIGNMENT, TRANSPORTATION, AND MINIMUM COST FLOW NETWORK PROBLEMS

by

D. Klingman*
A. Napier**
J. Stutz***

February 1973

* Associate Professor of Operations Research at the University of Texas, Austin, Texas.
** Research Associate, Operations Research Division, Continental Oil Company, Houston, Texas.
*** Assistant Professor of Operations Research at the University of Texas, Austin, Texas

This research was partly supported by a grant from the Farah Foundation and by ONR Contracts N00014-67-A-0126-0008 and N00014-67-A-0126-0009 with the Center for Cybernetic Studies, The University of Texas. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 512
The University of Texas
Austin, Texas 78712

DDC
RECEIVED
APR 6 1973
B

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Center for Cybernetic Studies
University of Texas at Austin

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

NETGEN - A Program for Generating Large Scale (Un)-Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(First name, middle initial, last name)*

D. Klingman J. Stutz
A. Napier

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
February 1973	28	22

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
NR-047-021	Center for Cybernetic Studies Research Report CS 109
b. PROJECT NO. N00014-67-A-0126-0008	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
c. d. N00014-67-A-0126-0009	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research(Code 434) Washington, D.C.

13. ABSTRACT

One purpose of this paper is to describe the development, implementation, and availability of a computer program for generating a variety of feasible network problems. In particular the code can generate capacitated and uncapacitated transportation and minimum cost flow network problems, and assignment problems. In addition to generating structurally different classes of network problems the code permits the user to vary structural characteristics within a class.

Since researchers can generate identical networks using this code, another purpose of the paper is to provide problems benchmarked on several codes currently available. In particular, the later part of the paper contains the solution time and objective function value on 40 assignment, transportation and network problems varying in size from 200 nodes to 8,000 nodes and from 1,300 arcs to 35,000 arcs.

DD FORM 1 NOV 68 1473 (PAGE 1)

S/N 0101-807-6811

I

Unclassified

Security Classification

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Transportation Problems						
Network Problem						
Transshipment Networks						
Assignment Problems						
Network Times						
Transportation Times						
Assignment Times						

1.0 COULD YOU USE THESE PROBLEMS?

The purpose of this paper is to describe the development, implementation, and availability of a computer program for generating a variety of feasible large scale L.P. problems which are generally termed network problems. Among the most frequently discussed network problems which the code can generate are capacitated and uncapacitated transportation and minimum cost flow (pure) network problems;as well as the simpler forms such as assignment, shortest path, and maximum flow problems. In addition to generating structurally different classes of network problems the code permits the user to vary structural characteristics within a class. The user controls the size of the problem as well as various parameters. In particular,the user controls the size by specifying the number of pure sources (origins), pure sinks (destinations), transhipment nodes with supply (transhipment source nodes), transhipment nodes with demand (transhipment sink nodes), transhipment nodes with no supply or demand (pure transhipment nodes), and the total number of arcs (cells) in the problem. The user has additional control of the structure through the use of such parameters as: total supply, cost range, upper capacity range (the lower capacity of an arc is always zero), percentage of arcs to be capacitated, and another parameter which implicitly controls solution difficulty. Another feature of the program is the inclusion of a random number generator [4,22] with a user supplied seed. This feature allows the code to regenerate the same problem if every input parameter is the same. Thus, if one researcher wants to solve the same problems that another has solved, he may do so by using the same input parameters. (Because of differences in word size and hardware arithmetic this may not be true across all computers.) Since researchers can generate identical networks, another purpose of the paper is to provide problems benchmarked on several codes currently available. While the set of benchmarked problems is small (40 problems) their characteristics are widely diversified.

2.0 MORE PROBLEMS - WHO NEEDS 'EM?

Since the earliest computational experiments with network computer codes both users and developers have been faced with the necessity of evaluating such codes as to their capacity, speed, effectiveness (with respect to problem structure), and accuracy. Initially most efforts were directed toward determining solution accuracy, however, more recently attention has been focused on the other attributes. This is due in large part to the availability of competing algorithms [1,2,3,6,8,10,11,12,13,14,15,16,17,19, 20], the development of larger network models, the increased use of network models in government and industrial applications, and new techniques for converting problems which otherwise appear to be unamenable to network formulation. Current computational studies [1, 2, 13, 14, 16, 20] have concentrated on problem size and solution time (relative to a particular class of networks) because of the interest of potential users to minimize their computer costs.[1/] Also some models are not being implemented in industry due to the prohibitive solution time and lack of computer codes to handle extremely large problems.[2/] Further interest in reducing solution time is stimulated by the fact that network problems often occur as subproblems in larger problems such as warehouse location problems, multi-commodity network problems, fixed charge transportation problems, and constrained transportation problems.

The problem of adequately "benchmarking" even the most thoroughly debugged codes arises, of course, in a variety of applications of computers to mathematical and scientific problems. However, many network problems involve quite large node-arc incidence matrices (say 1500 by 10,000; that is, 1500 nodes and 10,000 arcs). Consequently, the data handling and generation of test problems become severe even for problems with well-known topological characteristics. For this reason, tasks like the comparison of performance of network codes

must be carried out with the raw material (i.e. test problems) at hand.

Thus, one of the areas wherein our contribution lies is that of providing yardsticks for the evaluation of a number of overall performance characteristics of reasonably well debugged codes. Perhaps one of the most useful applications of our code is that of measuring the solution time and accuracy of some well-known and widely used network codes when employed to solve very large problems (e.g.,the code could easily be used to create 10,000 node problems of any desired arc density). Further, since this generator can be used to recreate problems, developers of new codes could use several standard "benchmarks" to compare their codes. Also, the availability, in quantity, of a meaningful variety of test problems, may help to influence the implementation of new solution techniques for network problems. All too often with new algorithms, an elegant theory has been a substitute for, rather than gone hand-in-hand with, effective performance in practice.

Some other attributes of having such a code available are:

1. To permit codes developed for a more general class of problems to be easily tested on special subclasses. For instance, codes developed to solve general minimum cost flow network problems could also be easily benchmarked on transportation and assignment problems, thus providing ways to evaluate the relative worth of the more specialized codes. To illustrate numerous algorithms exist for solving transportation problems and minimum cost flow network problems. Theoretically these problems are equivalent since any minimum cost flow network problem can be reformulated as a transportation problem. However, the question arises, "Is it worth developing and maintaining separate codes for each of these problem types, or should only one of the codes be developed? If only one, which type should be developed?" From a theoretical standpoint, the O.R. literature reflects the feeling that both types of algorithms should be pursued. Similar

questions are relevant between algorithms for these problems and assignment problems or maximum flow problems.

2. To encourage standardization of data specification for all types of network problems. One of the problems we encountered in trying to benchmark codes based on different methodologies (e.g., codes based on circulation networks such as out-of-kilter [2,3,17] and codes based on the simplex method [6,12,13,14,20] and codes designed to solve different types of problems was their lack of uniformity for input specification. To illustrate, out-of-kilter implementations assume that the input will be an arc at a time and that the network is a circulation network. The simplex based codes assume that the initial inputs will be the supply or demand of each node followed by the arcs; however, even this is not standard, since some transportation codes assume that the supplies and demands are followed by the cost matrix and upper capacity matrix (if the problem is capacitated). Within this framework some codes assume the input of a complete cost matrix while others assume the input is by origin with a cost and destination node number for each admissible cell in the problem. To add more confusion and frustration some of these codes assume the destinations are numbered starting at 1 and others from the number of origins plus one. This non-standardization of problem specification (in terms of input format) is most frustrating and has hampered benchmarking since researchers are reluctant to re-code their input routines. Thus it is essential to establish a standard way of specifying all types of network problems as well as network problems within a class. In order to achieve this standardization with minimum user inconvenience, we use a network specification which is compatible with SHARE [3,17] out-of-kilter since this is probably the most widely used network format. (See Appendix.)

3. To facilitate computational studies on the effect of parameter variation--such as changing the cost range, total supply, percentage of arcs capacitated, number of arcs, and capacity range, etc.

4. To generate problems which will test various parts of a code. Since all the problems generated by this code are feasible, it is clear that we can not generate problems capable of testing all parts of network solution codes. In general, the validation phases in the construction of large scale network codes present numerous challenges for adequately testing all parts of the code. While establishing problems which will accomplish this is virtually untracked territory, we have tried to design this code to generate problems which will provide various challenges to solution codes. For example, the code is capable of generating different network topologies (e.g., assignment, transportation, and general network problems) with differing characteristics within each problem type. Thus good programming practice and the use of this generator can help to avert future fiascos, similar to those which are undocumented but which have become legend in the folklore. (An early example is the L.P. "nut mix code", so called because of its ability to speedily solve this textbook problem, which had been used as a test problem during code development, and to solve no other problems .)

3.0 The Creation Process

With these thoughts in mind we turn our attention to a more specific description of the methodology and other salient features of the generator. Having read the input parameters (which are described in the Appendix) the size, type, and characteristics of the problem are fixed. The program then creates a network problem within this framework.

The overall program can be divided into two main parts. The first part creates what is called a skeleton network and is concerned with obtaining the proper number of nodes of each type, insuring the correct total supply, and guaranteeing that the resulting problem will be connected and feasible. The second part completes the problem while insuring that the remaining specifications are met, such as total number of arcs, cost range, upper bound range, and percentage of arcs capacitated.

First, all nodes are given a number (integer) between one and the number of nodes, and the nodes are grouped into sets by type (i.e., pure source, transhipment source, pure transhipment, pure sink, and transhipment sink). During this part of the program, transhipment sources and sinks are treated as pure sources and sinks, respectively. The total supply is then randomly distributed among the sources as follows: (The program uses random numbers from a uniform probabidity distribution [4,22].) the total supply is divided by the number of sources and each source is initially assigned the integer part of this amount. Subsequently, the amount assigned to each source is randomly split into two integer amounts. Each source retains one part of the split and the other part is assigned to a source chosen at random. The supply of each source is, thus, equal to the sum of the parts assigned to it. Since the intital division of the total supply was truncated to equal integer portions, any unassigned supply is also assigned to a randonly chosen source.

Next, a tree is randomly created from each source node by generating arcs involving the source node and a random number of pure transhipment nodes. These trees (called chains due to their structure) are pairwise disjoint and mutually exhaustive of all nodes except sinks. In each chain

there is a single directed path from the source node to every pure transhipment node in the chain.

After the chains have been created, each chain is connected to a random number of randomly chosen sinks. During this procedure the last node in each chain is always connected to a sink while the remaining sinks picked for a chain are connected to members of the chain chosen at random. Simultaneously, the demand amounts are accumulated for each sink by randomly distributing the supply of the unique source among the sinks connected to the chain, in a manner analogous to the distribution of the total supply. Since two or more chains may be connected to the same sink, the demands of the sinks are successively accumulated as new chains are attached.

In the case of transportation and assignment problems, since they contain no pure transhipment nodes, each chain contains only a source (origin) node. For transportation problems the origins are connected directly to a random number of sinks (destinations) and the demands are created by distributing the supply as above. For assignment problems each origin is randomly connected to a unique destination and each origin is given a supply of one and each destination is given a demand of one.

At this point the network has the correct number of sources, sinks, transhipment nodes, and total supply. Also the network is guaranteed to be connected and feasible (without capacities). This partial network is called the skeleton and its generation completes the first part of the program. Observe that the skeleton will only contain a few more arcs than is required to have a connected graph. Thus, one of the major attributes of this procedure is its ability to create networks of extremely low density. It should be noted, however, that problems with extremely low density will possess similar

structural features due to this process.

The second part of the program begins by determining the costs and capacities for the arcs in the skeleton. Using the percentage of arcs to be capacitated (supplied in the input), certain arcs in the skeleton are randomly chosen to be capacitated. The upper capacity of each randomly chosen arc is set equal to the larger of the supply of the unique source of the chain in which the arc appears and the user supplied minimum upper bound. The remaining arcs in the skeleton are left uncapacitated. (Note that the capacity of arcs in the skeleton may be higher than the largest upper capacity supplied by the user.) A percentage (one of the input parameters) of the arcs in the skeleton (the specific ones are chosen randomly) are assigned the maximum cost. Other arcs are assigned a cost randomly chosen between the lower and upper limits. The flexibility to set the costs of a percentage of the skeleton arcs large is intended to discourage the use of these arcs in an optimal solution, thus, (possibly) making the network more difficult to solve. Figure 1 contains the skeleton generated by the code using the input specifications contained in Table I.

ITEM	NODES	SOURCES	SINKS	ARCS	COST RANGE MIN	COST RANGE MAX	TOTAL SUPPLY	TRANSHIPMENT SOURCES	TRANSHIPMENT SINKS	PERCENT HI COST	PERCENT CAPAC	UPPER BOUND RANGE MIN	UPPER BOUND RANGE MAX	RANDOM NO SEED
QUANTITY	14	3	4	32	10	100	75	1	2	35.000	70.000	100	400	123456789

Table I

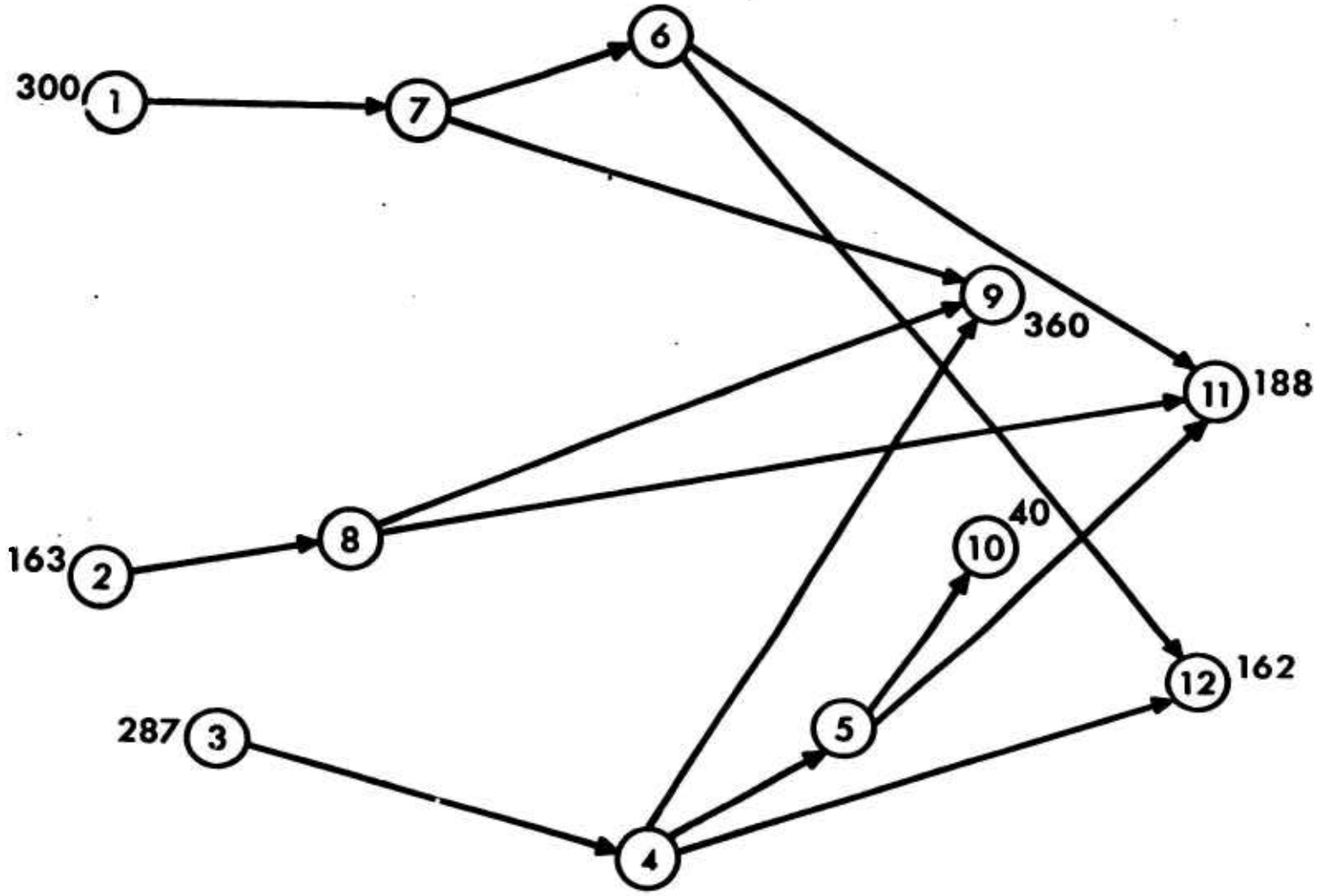

Figure 1 - Skeleton Created For The Specifications in Table I.

Once the skeleton is complete, the problem is guaranteed to be feasible regardless of the number, location, or characteristics of any additional arcs. Therefore, all that remains is to randomly generate and distribute the required number of additional arcs. (It is during this phase of the program that arcs are permitted to emanate from a transhipment sink and terminate at a transhipment source). For each node, except pure sink nodes, a random number of additional arcs are created from this node to other randomly selected nodes, while insuring that no duplicate arcs are created. The number of nodes to which a given node is connected, is randomly chosen from a range selected such that the final network will contain approximately the correct number of arcs. This range is dynamically adjusted, as each node is successively considered, in order to reflect the number of nodes remaining for consideration and the number of arcs remaining to be created. During this process a cost is randomly chosen (within the proper range) for each arc created and the given percentage of them are given an upper capacity randomly chosen within its range. This essentially completes the network except for adding a super-source (numbered one larger than the total number of nodes), and a super-sink (numbered two larger than the total number of nodes) in order to circularize the network. (The nodes and the arcs added for the purpose of creating a circulation network are not counted in the total number of nodes and arcs. They are inserted for compatibility with SHARE [3,17] input format only.) Figure 2 illustrates the final network completed from the skeleton appearing in Figure 1. (The dashed lines in Figure 2 correspond to the arcs added in order to circularize the network. The dotted lines correspond to the skeleton arcs.) The costs and capacities of each arc are given in Table II in the same format as they appear on the problem file. (See Appendix.)

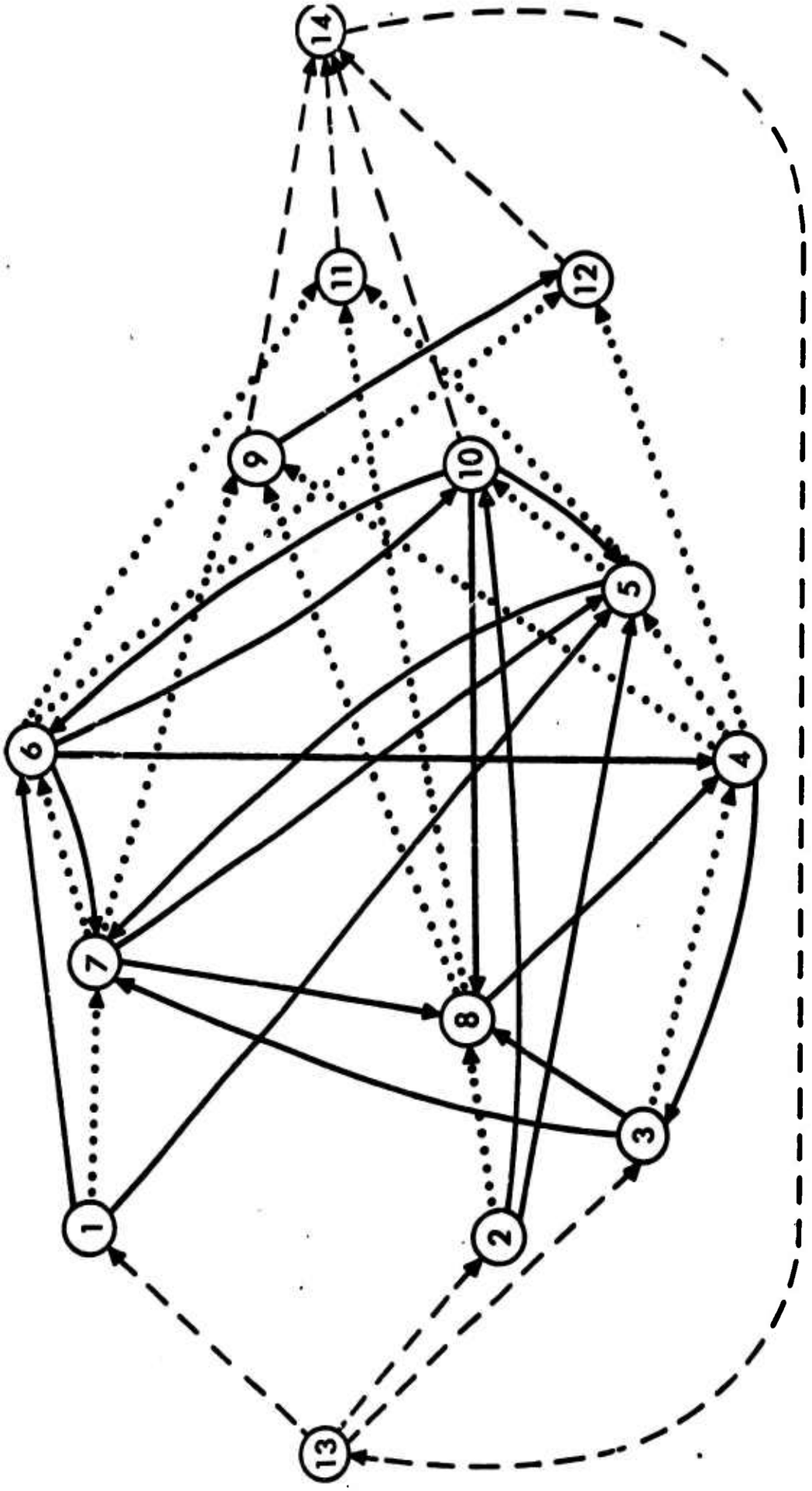

Figure 4 - Completed Network

TABLE II - Arc Parameters

FROM NODE	TO NODE	COST	UPPER CAPACITY	LOWER CAPACITY
13	1	0	300	
13	2	0	163	
13	3	0	287	
1	7	29	300	
1	5	93	750	
1	6	16	100	
6	12	20	750	
6	11	100	300	
6	10	25	100	
6	7	89	750	
6	4	24	100	
7	6	45	300	
7	9	85	750	
7	5	93	100	
7	8	74	750	
2	8	100	163	
2	10	11	100	
2	5	93	100	
8	11	35	163	
8	9	44	163	
8	4	27	100	
3	4	65	287	
3	8	93	100	
3	7	99	100	
4	5	21	287	
4	9	63	750	
4	12	27	287	
4	3	91	100	
5	11	47	750	
5	10	81	287	
5	7	39	100	
9	14	0	360	
9	12	65	100	
10	14	0	40	
10	5	32	100	
10	6	78	750	
10	8	56	100	
11	14	0	188	
12	14	0	162	
14	13	0	750	750

Note that all arcs have an upper bound, however those arcs which are considered uncapacitated have their upper bound set equal to the total supply. Also, observe that all arcs have a lower capacity of zero (blank field), except the arc connecting the super-sink to the super-source which has both upper and lower bound set to the total supply.

One of the more salient features of the code and the one chiefly responsible for its ability to create large networks is the way the network is stored. The skeleton is the only part of the network stored in core. Further, it is stored as a set of linked lists, each list corresponding to a chain. The demand for each sink and all of the arcs in the skeleton, except for the arcs connecting the chains to the sinks, are contained in one node length array. (All additional arcs are output to the problem file as they are created.) There are three other arrays (each one is at most node length in size) whose sole purpose is to insure non-duplication of arcs. The program contains only two other arrays. One contains the supplies of each source; the other is an array whose size is equal to the number of sinks and is used in connecting chains to sinks. Thus, the total array core requirements of the code is at most 5 times the number of nodes,which accounts for our ability to generate problems with thousands of nodes and an unlimited number of arcs. This is in contrast to the rudimentary network generators which use a node x node incidence matrix. The improvement in the size capability is in addition to the flexibility and generality discussed earlier. Thus, the generator should prove to be quite helpful to practitioners and researchers for many years since no existing codes are capable of solving such large networks.

4.0 Benchmarks

To help establish a small set of problems which researchers can use to benchmark their codes, we have generated and solved 40 network problems using 4 network codes. Table III contains the specifications of these 40 problems as required on the input cards. Problems 1-5 are 100 x 100 transportation problems; problems 6-10 are 150 x 150 transportation problems. The parameter specifications of the first ten were picked to correspond with the problems in [1,13] to provide a basis for comparison.[3] Problems 11-15 are 200 x 200 assignment problems.

Problems 16-27 are 400 node capacitated network problems; problems 28-35 are uncapacitated 1000 and 1500 node network problems. The parameter specifications of these problems, like the transportation problems, were picked to correspond with the problems in [1]. (The problems in [1] were generated using a preliminary version of this network generator.)

To facilitate and encourage the development of large scale codes,we have included a few problems (problems 36-40) which are at the frontier of large scale solution code capability. These problems are small, however, compared with the capability of the network generator.

The four codes which we used to solve the problems are those of SHARE [3,17], Boeing, SUPERK [1], and PNET. The first three codes are out-of-kilter codes while the last code is the special purpose simplex network code referenced in footnote 1f. The Boeing code, which was obtained through Chris Witzgall, was developed at the Boeing research laboratories. Table IV contains the solution times (not including input and output) and optimal objective function value for each of the problems. (The objective function values are included to help code developers verify the solution accuracy of their codes.)

Table III
PROBLEM SPECIFICATIONS

	Number of Nodes	Number of Sources	Number of Sinks	Number of Arcs	Cost Range Min	Cost Range Max	Total Supply	Transhipments Sources	Transhipments Sinks	Percent of High Cost	Percent of Arcs Capacitated	Upper Bound Range Min	Upper Bound Range Max	Random No. Seed
1	200	100	100	1300	1	100	100,000	0	0	0		0	0	13502460
2	200	100	100	1500	1	100	100,000	0	0	0	0	0	0	13502460
3	200	100	100	2000	1	100	100,000	0	0	0	0	0	0	13502460
4	200	100	100	2500	1	100	100,000	0	0	0	0	0	0	13502460
5	200	100	100	2500	1	100	100,000	0	0	0	0	0	0	13502460
6	300	150	150	4500	1	100	150,000	0	0	0	0	0	0	13502460
7	300	150	150	6500	1	100	150,000	0	0	0	0	0	0	13502460
8.	300	150	150	5155	1	100	150,000	0	0	0	0	0		13502460
9.	300	150	150	6075	1	100	150,000	0	0	0	0	0	0	13502460
10.	300	150	150	6300	1	100	150,000	0	0	0	0	0	0	13502460
11.	400	200	200	1500	1	100	200	0	0	0	0	0	0	13502460
12.	400	200	200	2250	1	100	200	0	0	0	0	0	0	13502460
13.	400	200	200	3090	1	100	200	0	0	0	0	0	0	13502460
14.	400	200	200	3750	1	100	200	0	0	0	0	0	0	13502460
15.	400	200	200	4500	1	100	200	0	0	0	0	0	0	13502460
16.	400	8	60	1306	1	100	400,000	0	0	30.	20.	16,000	30,000	13502460
17.	400	8	60	2443	1	100	400,000	0	0	30.	20.	16,000	30,000	13502460
18.	400	8	60	1306	1	100	400,000	0	0	30.	20.	20,000	120,000	13502460
19.	400	8	60	2443	1	100	400,000	0	0	30.	20	20,000	120,000	13502460
20.	400	8	60	1416	1	100	400,000	5	50	30.	40.	16,000	30,000	13502460
21	400	8	60	2836	1	100	400,000	5	50	30.	40.	16,000	30,000	13502460
22	400	8	60	1416	1	100	400,000	5	50	30.	40.	20,000	120,000	13502460
23	400	8	60	2836	1	100	400,000	5	50	30.	40.	20,000	120,000	13502460
24	400	4	12	1382	1	100	400,000	0	0	30.	80.	16,000	30,000	13502460
25.	400	4	12	2676	1	100	400,000	0	0	30.	80.	16,000	30,000	13502460
26	400	4	12	1382	1	100	400,000	0	0	30.	80.	20,000	120,000	13502460
27.	400	4	12	2676	1	100	400,000	0	0	30.	80.	20,000	120,000	13502460
28.	1000	50	50	2900	1	100	1,000,000	0	0	0	0	0	0	13502460
29	1000	50	50	3400	1	100	1,000,000	0	0	0	0	0	0	13502460
30.	1000	50	50	4400	1	100	1,000,000	0	0	0	0	0	0	13502460
31.	1000	50	50	4800	1	100	1,000,000	0	0	0	0	0	0	13502460
32.	1500	75	75	4342	1	100	1,500,000	0	0	0	0	0	0	13502460
33.	1500	75	75	4385	1	100	1,500,000	0	0	0	0	0	0	13502460
34.	1500	75	75	5107	1	100	1,500,000	0	0	0	0	0	0	13502460
35.	1500	75	75	6230	1	100	1,500,000	0	0	0	0	0	0	13502460
36.	5000	200	1000	15000	1	100	4,000,000	100	300	0	0	30	0	13502460
37.	5000	150	800	23000	1	100	4,000,000	50	100	0	0	0	0	13502460
38	5000	125	500	35000	1	100	4,000,000	25	50		0	0	0	13502460
39	5000	180	700	15000	1	100	4,000,000	100	300	1	0	3000	5000	13502460
40	8000	100	400	23000	1	100	4,000,000	50	100	1	0	4000	6000	13502460

TABLE IV

Solution Times (sec) and Optimal Objective Function Value

Problems	PNET	SUPERK	SHARE	Boeing	Objective Function Value
1	1.30	5.68	17.76	30.25	2,153,303
2	1.49	6.47	21.34	21.59	1,950,881
3	1.94	6.87	26.16	31.47	1,565,928
4	1.64	6.57	25.13	36.47	1,462,732
5	1.88	6.77	30.97	47.73	1,342,058
6	3.55	11.05	46.40	46.64	2,302,477
7	4.06	12.86	65.92	113.12	2,046,034
8	4.72	13.69	81.00	175.10	2,155,354
9	4.80	13.40	81.21	186.99	1,775,454
10	5.88	14.13	84.24	184.75	2,145,687
11	3.52	6.44	19.93	30.39	4563
12	4.87	6.47	21.17	22.08	3389
13	5.52	7.25	25.81	20.02	3070
14	6.02	6.95	24.95	23.11	2754
15	6.50	7.56	27.05	21.08	2721
16	2.40	5.27	21.51	15.05	69,612,156
17	3.11	8.36	32.40	64.64	46,831,850
18	1.92	5.13	20.06	18.31	68,197,261
19	3.60	8.49	31.75	61.07	45,816,193
20	2.67	4.69	18.11	25.72	65,940,530
21	2.76	7.96	32.60	61.39	48,575,656
22	2.22	4.60	17.91	24.84	65,770,640
23	3.00	7.91	32.66	67.96	48,503,656
24	3.12	5.59	25.27	21.57	92,612,577
25	4.17	8.37	33.19	48.40	60,418,740
26	4.45	5.51	25.05	19.34	82,612,189
27	4.42	7.50	30.45	41.98	56,665,337
28	6.35	13.91	53.87	83.98	122,582,531
29	7.39	14.51	52.55	117.83	105,050,119
30	9.08	16.00	61.33	152.21	86,331,458
31	9.59	17.05	61.33	135.73	82,561,499
32	15.70	22.88	78.63	553.93	174,279,219
33	20.20	25.89	101.92	210.14	195,931,070
34	17.10	25.42	92.25	248.16	160,007,929
35	19.39	29.96	DNR	DNR	162,270,303
36	384.081	NA	NA	NA	860,372,467
37	245.404	NA	NA	NA	351,773,733
38	140.982	NA	NA	NA	85,886,945
39	193.426	NA	NA	NA	512,111,082
40	105.097	NA	NA	NA	130,366,883

NA - Code and data would not fit in 104,000 words of memory.

DNR- Did not run.

The times reported in Table IV were obtained on a CDC 6600 using the FORTRAN RUN compiler. The authors have solved some of these problems on a UNIVAC 1108 using the FORTRAN V compiler, and on the IBM 360/65 using the H compiler. The times on the UNIVAC 1108 were about 10% slower, while the times on the IBM 360/65 were about 12% slower.

A noteworthy feature of the computational results that pervades the entire study is that PNET and SUPERK are decidedly superior to the other codes. (Roughly, PNET and SUPERK are at least 4 times faster and in many cases 8-10 times faster than the other codes.) Furthermore PNET strictly dominates SUPERK. (PNET is roughly twice as fast as SUPERK.) This is a surprising result since a non-extreme point algorithm is generally believed to be faster especially on assignment problems. Another advantage of the primal simplex approach indicated by the computations, is the core requirements of such a code. Whereas all of the out-of-kilter codes require at least 7 arc length arrays and 4 node length arrays, PNET only requires 3 arc length and 8 node length arrays. Thus PNET is capable of solving much larger problems than the other codes.

One of the unique findings obtained by a joint analysis of the three problem types is that assignment problems appear the easiest to solve, followed by general minimum cost flow network problems, and hardest to solve are the transportation problems. This finding is unexpected since it is a part of the folklore that transportation problems are easier to solve than network problems.

Another aspect of these computational results is that the transportation problem solution times on the out-of-kilter codes (particularly the SUPERK times) are substantially longer than (twice as long as) those reported in [1,13]. This result demonstrates the need for researchers to use a standard problem generator

since the same problem parameters were used to generate both the problems in [1,3] and these problems with different generators. Similarly the network problems 16-35 were generated using the same parameters and only different versions of our generator. However, the solution times reported in Table IV are slightly longer than those in [13].

Footnotes

1. Network codes and computational studies which are currently in progress include:

   a) a new out-of-kilter approach by M. Florian, department of Information, Universite De Montreal.

   b) a special purpose primal simplex network code by Graves, and R. Mc Bride , Graduate School of Business, University of California at Los Angeles.

   c) a minimum cost flow network code (the exact methodology to be employed is not known) by Burroughs Corporation.

   d) a new out-of-kilter approach by the Texas Water Development Board.

   e) a code for solving transportation problems with concave cost functions by R. Soland, Graduate School of Business, University of Texas at Austin.

   f) a computational study on a primal simplex network code by Glover, Karney, Klingman, Graduate School of Business, University of Texas at Austin.

2. To illustrate, Paul Randolph at New Mexico State University, in conjunction with the department of Agriculture has developed a 320 origin by 2250 destination transportation model with 56,000 admissible cells and twenty fixed charge variables for scheduling cotton to gins. The Center For Cybernetic Studies (directed by A. Charnes) at the University of Texas at Austin has developed a 400 origin by 1500 destination transportation model with 20,000 admissible cells and 3 extra constraints for scheduling slack production at Farah Manufacturing Corporation. Also the Center for Cybernetic Studies is developing a funds flow model for General Motors with 10,100 origins, 10,000 destinations and 500,000 admissible cells.

3. The problems in [1] were also used in the forthcoming computational study discussed in f of footnote 1.

References

1. Barr, R. S., F. Glover, and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," C. S. report 102, Center for Cybernetic Studies, University of Texas, BEB-512, Austin, 1972.

2. Bennington, G. E., "An Efficient Minimal Cost Flow Algorithm," O.R. Report 75, North Carolina State University, Raleigh, North Carolina, June 1972.

3. Clasen, R. J. "The Numerical Solution of Network Problems Using the Out-of-Kilter Algorithm," RAND Corporation Memorandum RM-5456-PR, Santa Monica, California, March, 1968.

4. Canavas, G. C. "GETRAN - Random Number Generator," NASA-Langley, 1967.

5. Dantzig, G. B. Linear Programming and Extensions, Princeton, N. J.: Princeton University Press, 1963.

6. Dennis, J. B. "A High-Speed Computer Technique for the Transportation Problem," Journal of Association for Computing Machinery, Vol. 8, (1958), 132-153.

7. Flood, M. M. "A Transportation Algorithm and Code," Naval Research Logistics Quarterly, 8 (1961), 257-276.

8. Florian, M. and M. Klein, "An Experimental Evaluation of Some Methods of Solving Assignment Problem," Can. Op. Res. Soc. Journal, 8 (1970), 101-108.

9. Ford, L. R., Jr., and D. Fulkerson, Flows in Networks, Princeton, N. J.: Princeton University Press, 1962.

10. Ford, L. R. and D. Fulkerson, "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," Naval Research Logistics Quarterly, 4, 1 (1957) 47-54.

11. Fulkerson, D. R. "An Out-of-Kilter Method for Solving Minimal Cost Flow Problems," J. Soc. Indust. Appl. Math, 9 (1961), 18-27.

12. Glickman, S., L. Johnson and L. Eselson, "Coding the Transportation Problem," Naval Research Logistics Quarterly, 7 (1960), 169-183.

13. Glover, Fred, D. Karney, D. Klingman, and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," To appear in Management Science.

14. Glover, Fred and D. Klingman, "Double-Pricing Dual and Feasible Start Algorithms for the Capacitated Transportation (distribution) Problem," University of Texas at Austin, 1970.

15. Grigoriadis, M. and W. Walker, "A Treatment of Transportation Problems by Primal Partition Programming," Management Science, 14, 9 (May 1968), 565-599.

16. Lee, S. "An Experimental Study of the Transportation Algorithms", Master's Thesis, Graduate School of Business, University of California at Los Angeles, 1968.

17. "Out-of-Kilter Network Routine," SHARE Distribution 3536, SHARE Distribution Agency, Hawthorne, New York, 1967.

18. Shamma, M.M. and T.A. Williams, "Constructing Sample Networks for Analyzing and Comparing Shortest Route Algorithms," 1972 ORSA-TIMS-AIIE Joint National Meeting Atlantic City, New Jersey.

19. Spivey, W.A. and R.M. Thrall, Linear Optimization, New York: Holt, Rinehart and Winston, 1970.

20. Srinivasan, V. and G.L. Thompson, "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," to appear in ACM.

21. Swart, W.W. "A Course Scheduling Problem: Model Development, Analysis, and Solution Algorithm," Presented to the 41st National Meeting of the Operations Research Society of America, New Orleans, Louisiana April 26-28, 1972

22. Tansworthe, R.C. "Random Numbers Generated by Linear Recurrence Modulo Two," Mathematics of Computation, 19 (1965) 201-209.

APPENDIX

INPUT-OUTPUT

Data Preparation

The data (input) deck must contain 2 cards (punched according to the format below) for each problem desired. The program will generate a separate network problem for each pair of cards in the data deck.

The following are the input requirements for each problem :

COLUMNS	CONTENTS	VARIABLE (TYPE)
Card 1.		
1-8	8 digit positive integer to initialize the random number generator. (Must have at least one non-zero digit in columns 1-3 and in columns 4-8 .)	ISEED(1) AND ISEED(2) (INTEGER)
Card 2.		
1-5	Total number of nodes	NODES (INTEGER)
6-10	Total number of source nodes (including transhipment sources)	NSORC (INTEGER)
11-15	Total number of sink nodes (including transhipment sinks).....	NSINK (INTEGER)
16-20	Number of arcs	DENS (INTEGER)
21-25	Minimum cost for arcs	MINCST(INTEGER)
26-30	Maximum cost for arcs	MAXCST(INTEGER)
31-40	Total supply	ITSUP (INTEGER)
41-45	Number of transhipment source nodes	NTSORC (INTEGER)
46-50	Number of transhipment sink nodes	NTSINK (INTEGER)
51-55	Percentage of skeleton arcs to be given the maximum cost	BHICST (REAL)
56-60	Percentage of arcs to be capacitated	BCAP (REAL)
61-70	Minimum upper bound for capacitated arcs..	MINCAP (INTEGER)
71-80	Maximum upper bound for capacitated arcs..	MAXCAP (INTEGER)

All input values on card 2 must be right - justified in their field.
The variables BHICST and BCAP should have a decimal point included.

To generate transportation and assignment problems the number of sources plus the number of sinks must equal the total number of nodes and the number of transhipment sources and sinks must be equal to zero (i.e., NSORC + NSINK=

NODES and NTSORC = NTSINK = 0). In addition, to create assignment problems the number of sources must equal the number of sinks and the total supply must be equal to the number of sources (i.e., NSORC = NSINK = ITSUP).

The maximum number of arcs which the program will create is equal to the number of pure sources (NPSORC = NSORC - NTSORC) times the remaining nodes (NONSORC = NODES - NPSORC) plus the total number of transhipments (NODES - NSORC+NTSORC-NSINK+NTSINK) times NONSORC -1. If the user asks for this number of arcs or greater the network produced will contain this number of arcs. (Note: A network containing this number of arcs is totally dense.)

Output Format

The program writes two files, an output file (printer) and a problem file (tape, disk or cards). The problem file contains all of the problems requested in a format compatible with SHARE [3,17] input format. Each problem consists of a set of card images written as follows (beginning in column 1):

```
BEGIN     (additional title information)
Title card for this problem
ARCS      (additional title information)
          arc data cards
END
SOLVE
```

Each of the arc cards is written as follows:

cols.	1-6	blank
	7-12	number of "from" node
	13-18	number of "to" node
	19-20	blank
	21-30	cost
	31-40	upper capacity
	41-50	lower capacity

Table II contains the arc data cards (as written on the problem file) for the problem specification given in Table I. (Note that all the arcs emanating from a node appear together.) The three header cards (BEGIN, title, ARCS) for this problem are given below:

```
BEGIN      RANDOM NUMBER GENERATOR SEED=12345678, PERCENT HI CST=35.00
NODES = 12,SORCE=   3,SINK=   4,ARCS=   32,COST=  10- 100,SUPPLY=  750
ARCS   TSORCE=   0,TSINK=    2,CAP=  100-    400,PERCENT CAP= 70.00
```

A file containing n problems will appear as follows:

```
BEGIN
title card
ARCS
          arc data cards                          } Problem 1
END
SOLVE
                .
                .
                .
BEGIN
title card
ARCS
          arc data cards                          } Problem n
END
SOLVE
QUIT                                    End of job
```

The output file consists of a summary of the user input specifications and the actual number of arcs generated for each problem. Table I is similar to the summary produced by the code.